ACTIVE NEURAL LEARNERS FOR TEXT WITH DUAL SUPERVISION

by

Chandramouli Shama Sastry

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
November 2018

Dedicated to parents, friends, roommates and relatives.

Table of Contents

List of Tables

List of Figures

Abstract

Dual supervision for text classification and information retrieval, which involves training the machine with class labels augmented with text annotations that are indicative of the class, has been shown to provide significant improvements, both in and beyond active learning (AL) settings. Annotations in the simplest form are highlighted portions of the text that are indicative of the class. They can range from unranked document-specific phrases to ranked corpus-level class-indicating terms and are an easy way to better engage users in training process. In this work, we aim to identify and realize the full potential of unsupervised pre-trained word embeddings for text-related tasks in AL settings by training Neural Nets – specifically, Convolutional and Recurrent Neural Nets – through dual supervision. The proposed solution involves the use of gradient-based feature attributions for constraining the machine to follow the user annotations; further, we discuss methods for overcoming the architecture-specific challenges in the optimization. Our results on the sentiment classification task show that one annotated and labeled document can be worth up to 7 labeled documents, giving accuracies of up to 70% for as few as 10 labeled and annotated documents, and shows promise in significantly reducing user effort for total-recall information retrieval task in Systematic Literature Reviews.

Acknowledgements

I am most deeply indebted to my advisor, Dr. Evangelos Milios. Highly supportive and encouraging since the beginning, he has helped me in every step through my masters program. He introduced me to tools like Mindmup and helped me wade through the literature, finally leading me into the right research topic. I couldn't have done this research without his insightful ideas and directions. He even advised me in the course-related projects and patiently taught me how to prepare presentations! Not only did he teach me how to write better, but also taught me how to write in LaTeX! I thank him for bearing with the countless messages I sent on Slack/E-Mail and his timely and detailed responses. He has also helped me in finding an internship and planning the way forward. He has helped me in getting through a number of hurdles in graduate study. I consider myself blessed to have got an opportunity to work under the direction of Dr. Evangelos Milios. I am ever grateful to him for funding and supporting my entire masters program.

I am grateful to my teachers – Dr. Stan Matwin, Dr. Thomas Trappenberg, Dr. Vlado Kesselj and Dr. Fernando Paulovich – for teaching Machine Learning, Natural Language Processing and Visual Analytics so well! The ideas gained from these classes have played a huge role in shaping my thoughts and have indirectly contributed to my thesis.

I would like to thank Dr. Sageev Oore for giving me an opportunity to function as the Teaching Assistant for the Deep Learning course and the subsequent opportunity to work under his direction at Vector Institute. I got exposed to a lot of interesting ideas in Deep Learning and helped in positioning my work.

Finally, I would like to thank my parents for instilling in me the values of higher education and supporting me through my ups and downs through daily skype calls. I am also grateful to my roommates – Mohanish Gunwant, Nikhil Dhirmalani and Rahul Midha – for bearing with the frustrations and stresses during the process of research. I am thankful to my friends – Darshan Jagaluru, Bharath Kashyap, Mohit Mayank, Sai Vishwas, Deepak Mahendrakar, Gautham BA, Aman Achpal, Karthik Radhakrishnan and Debarati – for keeping in touch and keeping me informed of the ongoings back home and never allowed me to feel left out.

Chapter 1

Introduction

Active Learning (AL) is an iterative learning process wherein the machine cleverly chooses data points to present to the human to elicit labels which are most helpful in inducing the best classifier. While the learning usually starts with a completely unlabeled dataset, the machine starts accumulating data through a sequence of carefully chosen queries. The goal of the learning algorithm is to induce the best classifier for a given budget – which is usually defined in terms of the user effort. Usually, the unlabeled pool is much larger than the labeled pool and algorithms which can make use of large amount of unlabeled data can benefit the most in the AL setting.

Distributed word representations induced from unlabeled text data, which capture several syntactic and semantic relationships between the words, have advanced the state-of-the-art in several natural language processing tasks such as Text Classification, Question-Answering, Named-Entity-Recognition. However, Convolutional Neural Networks and Recurrent Neural Networks which can make the most effective use of these word embeddings haven't been explored much in the Active Learning setting; the main reason is that the ideal decision boundaries can be found only when constrained with a lot of training examples. In this work, we propose to better exploit unsupervised embeddings by the use of user annotations. Although less pronounced, this problem exists in traditional machine learning with linear decision boundaries as well and has been usually solved by eliciting user annotations (Zaidan et al. [2007], Melville and Sindhwani [2009], Small et al. [2011], Sharma et al. [2015]).

Annotations are essentially feature labels which indicate whether or not the feature is indicative of the class assigned to the containing document. Some examples of annotations for sentiment classification of movie reviews, as reproduced from Zaidan et al. [2007], are:

- **you will enjoy the hell out** of American Pie
- fortunately, they **managed to do it in an interesting and funny way**.

- he is **one of the most exciting martial artists on the big screen**, continuing to perform his own stunts and **dazzling audiences** with his flashy kicks and punches.
- the romance was **enchanting**.

How are annotations used? During training, the machine implicitly learns to select useful features from input and associates patterns in selected features with classes; however, when not constrained with lot of instances, the machine has a wide variety of choices which do not necessarily generalize well on unseen instances. The choices the machine makes and the importance of each of the input dimensions in computing the class assignment are represented as attribution maps; an overview of gradient-based attribution methods can be found in Ancona et al. [2018]. Annotations are used to suggest the useful choices.

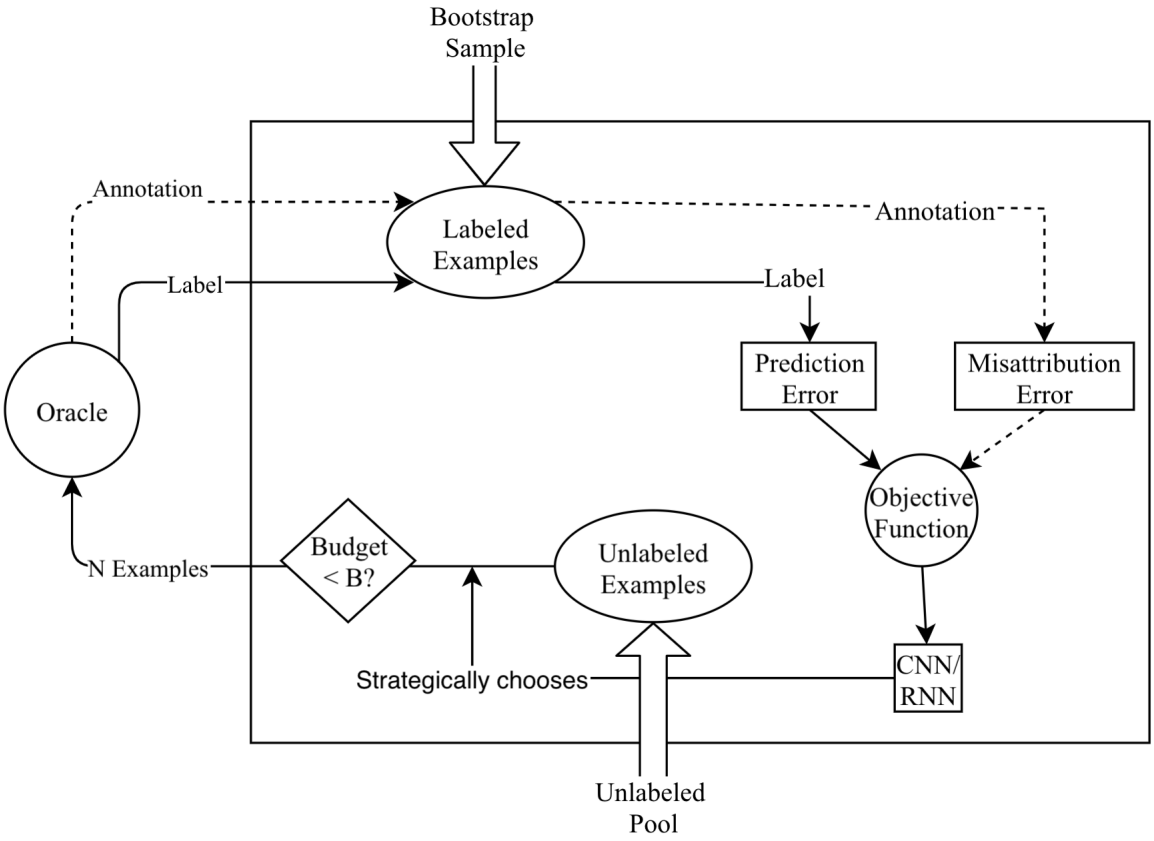

Figure 1.1: Overview of active learning: The process starts with a sample of labeled and annotated instances. After every training round, N instances are strategically selected from the unlabeled pool and presented to the human oracle for labeling (and annotating); these are repeated until the budget B (measured in terms of instances or time) is exhausted. In dual supervision, the system is constrained by the additional misattribution error – which quantifies the deviation between machine-selected features and user annotations. In our experiments, we simulate the oracle through code.

In this work, we propose a method for training neural networks by factoring in user annotations. We define misattribution error as the deviation between machine-selected features and user annotations; the core of the paper involves defining misattribution error. Thus, the strategy for factoring in user reviews involves minimizing misattribution error along with the prediction error as shown in Fig. 1. Note that, we query a batch of data points per iteration as opposed to one data point per iteration.

We perform our experiments [1] on (i) the sentiment classification task and (ii) total recall information retrieval (requiring the collection of all relevant documents) in Systematic Literature Reviews in Software Engineering. We attempt to answer the following questions:

1. How can we factor in annotations in training Convolutional Neural Networks and Recurrent Neural Networks for the text classification task?

2. What is the return on investing additional time for annotating each instance in addition to labeling it? Experimentally, the time to annotate an instance has been found to be at most be equal to the time taken to label the instance itself (Zaidan et al. [2007]).

3. Can the neural nets model user-selection well enough that the extra investment on annotation can be minimized? Unlike bag-of-words model, the machine can automatically estimate importance of unseen words and can considerably reduce annotation costs; further, reducing annotation costs need not use specialized sampling strategies like the ones in Melville and Sindhwani [2009] and Sharma et al. [2015].

[1] Code for replicating the experiments: `www.github.com/chandramouli-sastry/dual-AL`

Chapter 2

Related Works

We review related works which apply active learning using neural networks and review the general idea of factoring in annotations.

Active Learning for Text applications using Neural Networks like CNNs or RNNs is a relatively unexplored area. While we did not come across any work making use of RNNs for Text Classification, CNNs for Text classification in Active Learning is studied in Zhang et al. [2017b]. They propose a new sampling strategy that chooses those documents whose word embeddings are *expected* to change the most if the true class labels were revealed. They employ the idea of Expected Gradient Length and report improvements in accuracies over entropy-based uncertainty sampling. In our work, sampling method is a controlled variable and improvement in sampling strategies would bring in similar improvements in both training settings with or without user annotations.

Factoring Annotations in Neural Networks using gradients has been proposed in Ross et al. [2017]. To the best of our knowledge, we are not aware of any work on factoring annotations in CNNs/RNNs. Although the approach we propose in our work draws inspirations from the ideas presented in this paper, there are a few weaknesses which make it inapplicable to Neural Networks with shared weights like CNNs and RNNs; we will discuss these weaknesses and how we overcome them in Section 4. While they factor in annotations implicitly, factoring in annotations explicitly using additional architectural component to model feature selection has been proposed in Zhang et al. [2016]. We note that this relies on longer annotations (sentences/phrases) and is not targeted to work well for shorter (words) annotations. Further, we are motivated towards finding a solution which can be applied to any given architecture requiring no architectural modifications.

Factoring Annotations in traditional models has been well explored – both within and beyond Active Learning. Annotations to improve SVM performance on movie sentiment reviews (Pang and Lee [2004]) are proposed in Zaidan et al. [2007]; they also studied the time it takes to annotate and report that annotating a document can take as much time

as that of labeling it. Further, this idea has been extended to Generative models in Zaidan and Eisner [2008] and Melville et al. [2009]. The idea of ranked feature annotations for Information Retrieval and Text classification using SVMs has been proposed in Small et al. [2011]. More recently, novel query strategies and a unified method for training SVMs, Logistic Regression and Naive Bayes augmented with annotations in an AL Setting are proposed in Sharma et al. [2015]; the idea is to modify the training instance by scaling down the irrelevant features. While this idea did not give us any improvements in CNNs or RNNs, adding a new instance with just relevant features helps in CNNs (when there are more training instances) and we show that it is a special case of what we propose (See Appendix A). Unfortunately, even this strategy did not work well for RNNs.

Posterior Regularization in machine translation (Zhang et al. [2017a]) shows how a variety of prior knowledge can be factored into the training of the LSTM as differentiable constraints. The annotations in our work correspond to the prior knowledge represented in phrase table and bilingual dictionary. In order to express these constraints, they make use of the learnt attentions; we would like to point out that attentions are not always explicitly modeled by an architectural component and needs to be derived using attribution techniques. Further, when we trained the RNN-architecture with constraints only on the learnt attention, the improvement was not significant.

Chapter 3

CNN and RNN Architectures

In this section, we will describe the architectures we used. As the goal of the paper is to study how annotations can be factored in, we use standard values without any hyper-parameter tuning.

3.1 CNN for Text Classification

We will use the CNN architecture proposed in Kim [2014] and investigated in detail by Zhang and Wallace [2015]. A simplified architecture is shown in Figure 3.1 (adapted from (Zhang and Wallace [2015] and Kim [2014])).

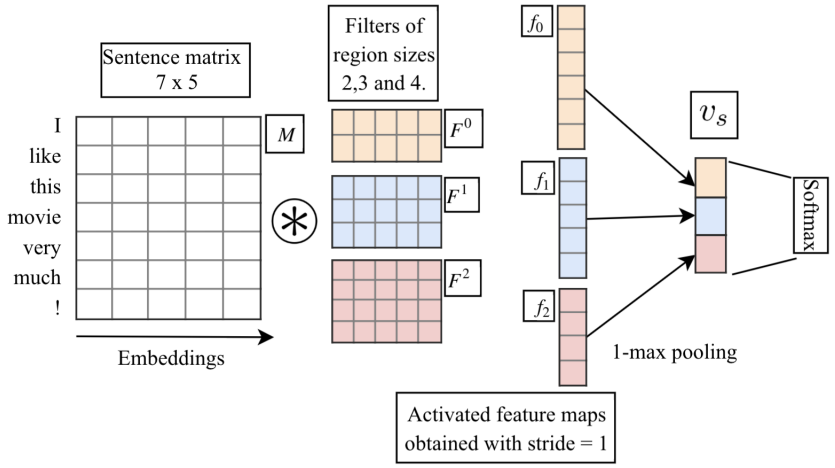

Figure 3.1: CNN Architecture: An input sentence is converted into a sentence matrix, M, of dimension $N \times d$ where N is the number of words and d is the dimension of word embedding. The activated feature maps (f_is) (Eq. 3.1) are computed by applying 1-D convolution over M using k filters (F^is) of sizes $r_i \times d$, where r_i, the region size, indicates the number of adjacent words jointly considered. The feature vector, v_s, is constructed by applying 1-max-pooling (Eq. 3.2) on each of the activated feature-maps; the dimensions of v_s are therefore equal to k. v_s is used as feature-set for softmax classifier.

Let us denote the i^{th} filter by F^i and its region size, the number of adjacent words jointly considered, by r_i. A filter bank of k filters will yield k activated feature maps; if we

denote the i^{th} feature map as f_i , the j^{th} activation (given $0 \leq j \leq N - r_i + 1$) is given by :

$$f_{ij} = \sigma\left(\sum_{p=0}^{p=r_i}\sum_{q=0}^{q=d} M_{p+j,q} \times F^i_{p,q}\right) \tag{3.1}$$

The i^{th} element of v_s is obtained by applying 1-max pooling over f_i:

$$v_{si} = \max_{0 \leq j \leq N-r_i+1} f_{ij} \tag{3.2}$$

We used 50 filters each of sizes 3,4 and 5 as suggested in Zhang et al. [2017b] and Zhang and Wallace [2015]; the weights of the filters and their biases and the weights and biases of the softmax are trainable parameters. However, we trained the machine with static embeddings as suggested by Zhang and Wallace [2015] (unlike Zhang et al. [2017b]). We applied dropout regularization to the penultimate layer (v_s) with dropout probability of 0.3. Dropout with probability 0.3 means that we *expect* 30% of the 150 dimensions to be set to zero.

3.2 RNN for Text Classification

We use an adaptation of the Hierarchical Attention Network Yang et al. [2016] for text classification; specifically, we use single GRU (Gated Recurrent Units) instead of bidirectional GRU and we ignore the hierarchy and consider the document as one long sentence. The architecture is shown in Fig. 3.2.

Let N_h be the number of hidden units, d be the number of dimensions in the embedding and w_i be the embedding representing the i^{th} word. The trainable parameters are represented by $W*$ and $U*$. The hidden states h_i are derived using standard GRU formulation (The operator $\odot$ denotes element-wise multiplication and [A:B] denotes the concatenation of vectors A and B):

$$h_i = (1 - z_i) \odot h_{i-1} + z_i \odot \tilde{h}_i \tag{3.3}$$

where, $z_i = \sigma(W_z^{N_h \times (N_h+d)}[w_i : h_{i-1}] + b_z)$

$r_i = \sigma(W_r^{N_h \times (N_h+d)}[w_i : h_{i-1}] + b_r)$

$\tilde{h}_i = f(W_h^{N_h \times d} w_i + r_i \odot (U_h^{N_h \times N_h} h_{i-1}) + b_h)$

where f is a non-linearity like tanh. The reset gate r_i and the forget gate z_i control the weighting between older information and newer information.

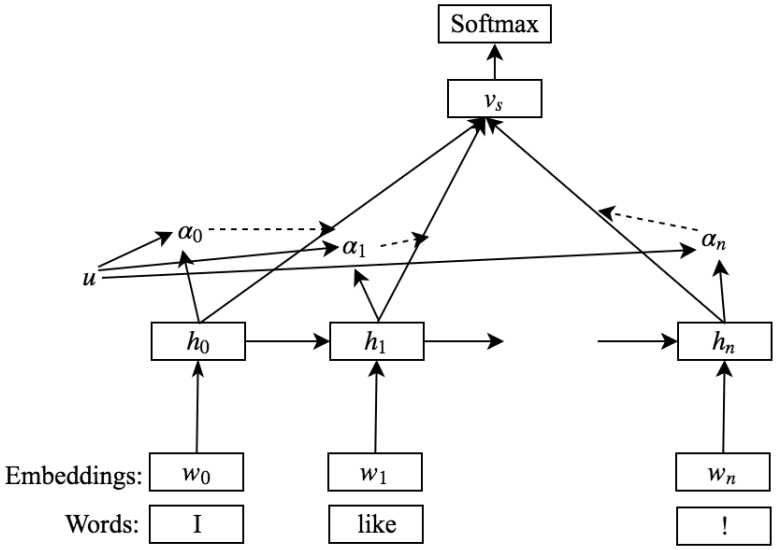

Figure 3.2: RNN Architecture: The input sequence w_i of words represented by their embeddings and the hidden states h_i (Eq. 3.3) are computed according to standard GRU formulation. The attentions α_i are derived using the context vector u (Eq. 3.4) and feature vector v_s is computed by taking the weighted mean of the hidden activations (Eq. 3.5). v_s is used as feature-set for softmax classifier. See Figure 3.3 for more details.

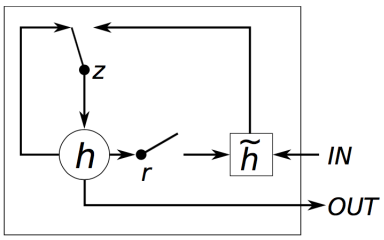

Figure 3.3: GRU unit: Illustration of Eq. 3.3 as taken from Chung et al. [2014]. IN corresponds to [h˙i-1:w˙i] and OUT corresponds to h˙i.

The attentions α_i are derived using context vector u Bahdanau et al. [2014] as :

$$\alpha_i = \frac{\exp(u_i^T u)}{\sum_j \exp(u_j^T u)} \tag{3.4}$$

where $u_i = \tanh(W_A h_i + b_A)$. The feature vector v_s is later obtained as:

$$v_s = \sum_i \alpha_i h_i \tag{3.5}$$

The trainable parameters include the various weights and biases used for computing h_i along with the attention parameters (W_A and b_A) and the context vector u. We used hidden units of size 100 and context vector of size 50. We found that the RNNs needed more regularization than CNNs and used all of the following: variational dropout between recurrent states (Gal and Ghahramani [2016]), input embedding dropout, activity regularization of recurrent states (Merity et al. [2017]) and L2 regularization.

3.3 Softmax

Once the feature vector v_s is obtained, we feed it through the softmax classifier parameterized by weight W_s and b_s and obtain the logits as:

$$l = W_s v_s + b_s \tag{3.6}$$

We denote the logit of the expected class c as l_c. The probability of the document belonging to class i is then obtained by applying softmax over l:

$$p_i = \frac{\exp(l_i)}{\sum_j \exp(l_j)} \tag{3.7}$$

Chapter 4

Dual Supervision in NNs

The strategy that we adopt to factor in the user annotations is to first devise a method for computing misattribution error – the deviation between user suggested features (through annotations) and machine-chosen features and then minimize it. Annotations can be classified based on different criteria. For example, they can be classified into binary or real (with higher numbers indicating greater class-association) depending on whether they are unranked or ranked respectively. They can also be classified into instance-level or class-level depending on whether annotations are elicited per instance or per-class. Generically, A_i^U refers to weight assigned to i^{th} word by the user; while this can be binary or real, we assume it to be binary for the purposes of this discussion. In this work, we consider unranked annotations at both instance-level (Text Classification) and class-level (Information Retrieval). The same idea can be extended to ranked annotations: for example, one straightforward way would be to allocate a higher share (explained below) to higher class-indicating phrases.

4.1 Attributions and Misattribution error

A very straightforward method for computing attributions is given by computing the partial derivative of the output with respect to the input and multiplying it with the input itself (Shrikumar et al. [2016]). Thus, a_{ij}, the attribution score of the j^{th} dimension of w_i, the i^{th} word vector can be given as:

$$a_{ij} = w_{ij} \frac{\partial F(\mathbf{V_0}, \mathbf{V_1} ... \mathbf{V_n})}{\partial V_{ij}} \Bigg|_{\mathbf{V_0}=\mathbf{w_0}, \mathbf{V_1}=\mathbf{w_1} ... \mathbf{V_n}=\mathbf{w_n}} \tag{4.1}$$

F is a function of n vectors and represents the CNN/RNN. $\mathbf{V_i}$ is the placeholder (argument) for the i^{th} word vector. Best results are obtained when F is set to yield:

- l_c, the logit of the expected class, OR

- $\sum v_s$, the sum of the learnt dimensions of the feature vector in the penultimate layer. The sum is needed to capture the overall impact of V_{ij}.

The attribution score of the i^{th} word can then be obtained by simply summing across all the dimensions:

$$A_i = \sum_j a_{ij} \tag{4.2}$$

It is usually interpreted as the first-order Taylor approximation of the change in outputs if all components of i^{th} word are set to 0. In the context of word vectors, we interpret the attribution of a given word in an input as the quantification of inclination between the word embedding vector and the gradient vector, which happens to be normal to the decision boundary, given that the other words are in the exact same order. It can be expressed as a dot product as shown:

$$A_i = \mathbf{w_i} \cdot \frac{\partial F(\mathbf{V_0}, \mathbf{V_1} ... \mathbf{V_n})}{\partial \mathbf{V_i}} \bigg|_{\mathbf{V_0}=\mathbf{w_0}, \mathbf{V_1}=\mathbf{w_1} ... \mathbf{V_n}=\mathbf{w_n}} \tag{4.3}$$

Observe that the values of A_i are real-valued and can be positive or negative; positive values indicate that the word is in favor of the output and negative values indicate the opposite. What we are really interested in are the relative magnitudes and not their real magnitudes. We use the intuition that the attributions measure the inclination between the word embedding and *normals* (gradient vector) in computing the normalized attributions using softmax as:

$$\tilde{A}_i = \frac{\exp A_i}{\sum_j \exp A_j} \tag{4.4}$$

$\tilde{A}_i$ can be interpreted as the share of the i^{th} word in determining the class. One simple way of defining the misattribution error, μ, is by allocating a certain share, ρ, to the user-suggested features:

$$\mu(A^U, \tilde{A}, \rho) = \left(\sum_i A_i^U \tilde{A}_i - \rho \right)^2 \tag{4.5}$$

We use softmax for normalizing as opposed to other possible techniques as we found that encouraging the inclinations between *normals* and non-annotated words to be as large as possible ($A_i < 0$) gives superior performance; however, the degrees of freedom of the machine in allowing smaller inclinations to unannotated words is inversely proportional to the value of ρ. Since there are no constraints on the value of $\tilde{A}_i$, the machine has complete

flexibility in choosing number of words - it might choose very few or most of the words. Therefore, we can add additional constraints limiting the maximum or minimum share it has to allocate to each of the words. Because limiting the minimum would make the machine less robust to longer annotations, we choose to limit the maximum ($ReLU(x) = max(0, x)$):

$$\mu(A^U, \tilde{A}, \rho, \tau) = \left(\sum_i A_i^U \tilde{A}_i - \rho \right)^2 + \sum_i ReLU(\tilde{A}_i - \tau) \tag{4.6}$$

Summarily, we introduce two hyperparameters (ρ and τ) and the total error that needs to be minimized can be written as:

$$Error = Prediction_{Error} + \mu \tag{4.7}$$

The greater the value of ρ, the greater the share allocated to user annotations. The lower the value of τ, the greater the number of features it chooses from the user annotations. However, higher τ values correspond to greater flexibility in choosing the number of features.

In this study, we compute the attributions $\tilde{A}_i$ by either considering the F (Eq. 4.1) to yield output logit of the expected class or the features fed into the softmax layer. We distinguish these by μ_c and μ_f respectively. Note that the attribution values for different classes are different when μ_c is considered and are the same when μ_f is considered.

While this is a simple strategy, we encountered architecture-specific challenges in training CNNs and RNNs with constraints (Eq. 4.6) on attributions. Specifically: in CNNs, the gradients with respect to the input words not selected in 1-max-pooling are zero and are not informative making the optimization hard; in RNNs, the gradients received by the non-attended words are small and affects optimization. The exact adaptations of the misattribution error for the architectures are given in the Appendix A.

4.2 Comparison with Previous work

Constraining the neural net by minimizing the absolute value of the gradients with respect to input has been shown in Ross et al. [2017]. The misattribution error defined therein can be adapted to this problem as (with some notational abuse):

$$\mu = \lambda \sum_i (1 - A_i^U) \left(\sum_j \left(\frac{\partial F}{\partial V_{ij}} \right)^2 \right) \tag{4.8}$$

They report qualitative experiments showing that constraining the model explanations works well. Unfortunately, when we applied it to the Convolutional and Recurrent Neural Networks, we got results which were very comparable to the results of the machine without annotations and sometimes worse. The reason for this behavior is that, in machines with shared weights, the derivative of the output with respect to the input is primarily a function of the weights and has much less or no dependence (when activation functions like ReLU, which have 0 second derivative are used) on the input and functions as a regularizer.

The authors set the value of $\lambda = 1000$; we note that the magnitudes of the derivatives get smaller with depth and using higher λ values are imperative. Unfortunately, as the same λ value is also used for weighting the importance between prediction error and misattribution and setting it is non-intuitive as the same knob controls 2 behaviors. We overcome the dependence on large λ values by optimizing based on the relative magnitudes. Further, our strategy has at least 2 more advantages: allows derivatives with respect to unannotated features to be negative as well; it allows for specifying the allocation of shares to the annotated features.

Chapter 5

Experiments – Text classification

In this chapter, we attempt to answer the following questions in the context of text classifications: What is the return on investing additional time in annotating? Do we need to annotate all the examples? How does our method compare with earlier benchmarks? How well does the machine generalize in identifying useful class-indicating features?

We conduct experiments on text classification using:

- **Pang and Lee [2004]** dataset consists of 1000 documents each of positive and negative classes. 1800 of those (900 positive and 900 negative) are annotated by Zaidan et al. [2007]. We use 10-fold stratified cross-validation such that 9 partitions are used as part of the training pool and the remaining partition (containing 180 documents) and the 200 unannotated documents are used as held-out test data.

- **Maas et al. [2011]** dataset consists of 25k documents each in train and test split, equally divided amongst the positive and negative classes. As the test data is large enough, we do not perform cross-validation.

As the data points are balanced, we will use accuracy as the evaluation metric. The experimental setup of Active Learning is as follows:

- **Bootstrap phase**: The CNNs (RNNs) with and without dual supervision are both given the same bootstrap sample consisting of 5 positive and 5 negative samples; care is taken to ensure that the bootstrap samples, training and testing partitions are identical.

- **Iterative phase**: Until the budget is exhausted, each of the machines is allowed to choose 10 documents, which it is most *uncertain* about, to elicit labels and/or annotations using entropy-based uncertainty sampling quantified as ($P(Y = c|x)$ is the predicted probability that the instance x belongs to class c):

$$Unc(x) = -\sum_{c \in C} P(Y = c|x) \log(P(Y = c|x)) \tag{5.1}$$

where C is the list of all classes.

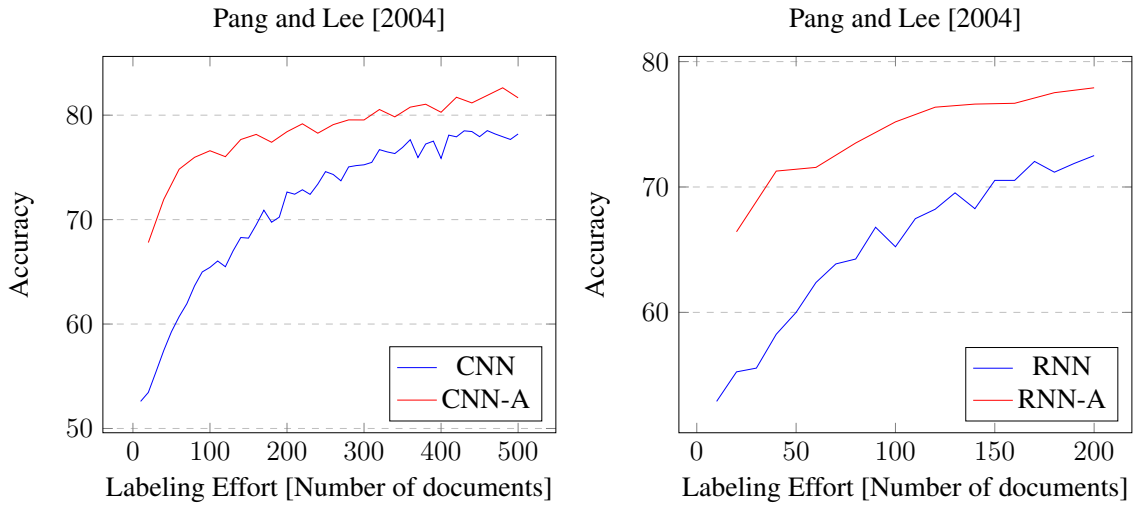

Figure 5.1: Learning Curves for dataset of Pang and Lee [2004]: The accuracy obtained using x annotated and labeled documents are compared with $2x$ labeled documents. While the vertical gap between the two curves indicates the gain in accuracy achieved for the same effort, the horizontal gap indicates the additional effort needed to achieve the same accuracy. We note that one labeled and annotated document can be worth up to 7 labeled documents in CNNs and up to 5 labeled documents in RNNs.

Every time the training data is updated, a new machine is instantiated and trained. The whole process is repeated 5 times with different seeds for computing partitions to accommodate for variance – in all, the accuracies are computed over 50 training sessions. For this task, we set the total share of user-suggested words, ρ, to 0.9 and the maximum share that can be allocated to a word, τ, to 0.8 (Refer to Appendix A.3 for a note on the hyperparameters). We used pre-trained Google News Word Vectors[1] for text-classification experiments. We aim to answer the following questions:

- **What is the return on investing additional time in annotating the instances?** In order to answer this, we will compare the accuracies of the machines with and without annotations for the same amount of time spent on labeling (Fig. 5.1). We can see that the CNN (RNN) augmented with annotations clearly outperforms CNN (RNN) with just labels even though the number of instances is double. The most interesting observation is that for as few as 10 labeled and annotated documents,

[1] `https://code.google.com/archive/p/word2vec/`

CNN achieves an accuracy of about 70% and without annotations, up to 150 labeled instances are required to match the accuracy! Similar observations hold true for RNN as well.

- **Do we need to annotate all the examples?** While the answer to this question might depend on the task in question, we seek to answer in the context of sentiment classification. In order to answer this, we design our experiments on Maas et al. [2011] dataset such that the dual supervised machine is bootstrapped using 10 randomly chosen annotated examples (5 from each class) from Pang and Lee [2004] dataset; in later rounds, only labeled examples (chosen from Maas et al. [2011]) are added to the training pool. The learning curve is shown in Fig. 5.2: we find that the machine can make good use of annotations even if a few of the documents are annotated.

- **How well does the machine generalize in identifying useful class-indicating features?** Assuming user annotations exhaustively indicate all class-indicating sentences/phrases, we investigate how well and how fast the machine (augmented with annotations) can mimic user annotations on unlabeled documents. For each of the documents not used for training, we compute the share assigned to user annotations, given by $\sum_i A_i^U \tilde{A}_i$, and examine the distribution of the assigned shares as the training progresses. Specifically: for CNNs, we compute $\tilde{A}_i$ using gradients as described; for RNNs, we use the attentions by setting $\tilde{A}_i = \alpha_i$. The results are shown in Fig 5.3. We observe that the machine can identify useful features quite well in the initial iterations of training process itself and can potentially help in conserving user effort. For example, the user can be shown the features that are selected by the machine and if they are good enough he can choose to just assign the class label.

- **How does this compare with earlier benchmarks?** Given that the Zaidan et al. [2007] dataset is quite popular and used in all the papers which aim to factor in annotations, we did not come across any paper which reported an accuracy of about 70% for just 10 documents. In order to quantitatively compare, we consider the relatively recent paper, Sharma et al. [2015], to be representative of the performances of the earlier benchmarks. As they report experiments on Maas et al. [2011] using a feature-expert, we executed their algorithm by bootstrapping using the annotated documents from the Pang and Lee [2004]. We find that this method was able to

achieve only about 55% for 10 labeled documents. Please refer to Appendix B for more results on standard text classification datasets.

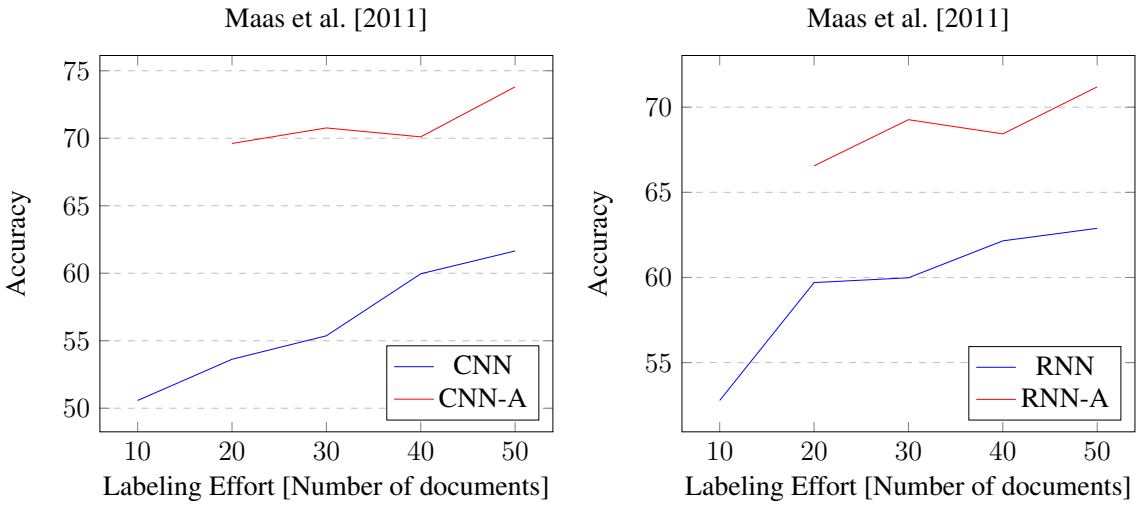

Figure 5.2: Learning Curves for Maas et al. [2011]: For this dataset, we bootstrap using annotated documents borrowed from Pang and Lee [2004] and use just labeled documents for later iterations. Therefore, we compare accuracy obtained using x labeled documents and 10 labeled and annotated documents with accuracy obtained using $x + 20$ labeled documents.

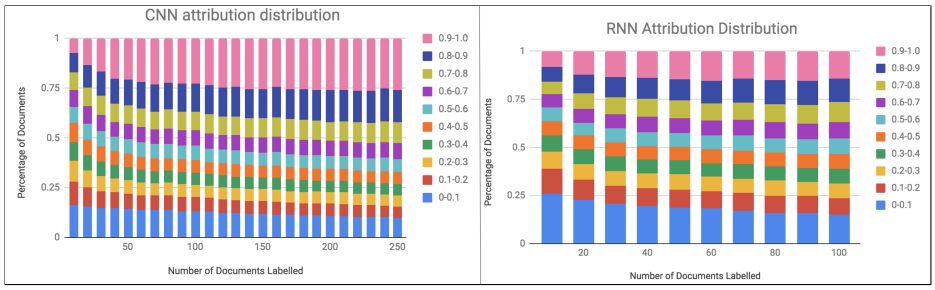

Figure 5.3: Attribution Distribution: The graphs show the distribution of shares (Eq. 4.4) allocated to class-indicating words in predicting the class as the training progresses. For CNNs trained with as few as 20 labeled and annotated documents, 50% of the unlabeled documents are assigned a share of at least 0.5 to the class-indicating words. The same can be observed with RNNs trained with 40 labeled and annotated documents.

Chapter 6

Experiments – Information Retrieval

In this section, we consider the task of primary study selection, requiring total recall information retrieval, in Systematic Literature Review (SLR). The initial steps of SLR involve defining a research question, extracting keywords, creating search strings and creating a database of all possible candidate studies from various data sources; this is followed by primary study selection which involves collecting all the relevant documents from the database and is one of the most time consuming processes requiring the review of thousands of documents to select a few dozen relevant documents.

In this work, we investigate how annotations can help in this task by applying it on 4 Software Engineering-based SLR datasets[1] corresponding to 4 SLRs as shown in Table 6.1, published and extensively studied by Yu et al. [2016]. The curated database essentially consists of the abstracts along with the relevance (class) label: 1 if they are included in the SLR, 0 otherwise. While the curated database along with relevance labels for Kitchenham and Brereton [2013] is reported to be shared by Dr. Kitchenham herself, the databases for the other 3 SLRs are reported to be constructed from IEEE xplore using search strings described in the corresponding papers and those studies which were included in the SLR are marked relevant. The experiment setup is as follows (adopted from Cormack and Grossman [2014] and Yu and Menzies [2017]):

1. The bootstrapping is done by selecting the top $N = 10$ documents according to BM25 score. This is repeated until at least one relevant and irrelevant documents are collected; fortunately, one bootstrap round was sufficient for all datasets. The details of the BM25 queries used for bootstrapping and number of relevant documents in bootstrap sample are given in the supplementary material.

2. The classifier is induced from the obtained labels (and annotations, if available). The reviewer is then presented with the top $N = 10$ most relevant documents (certainty sampling), as predicted by the classifier, for eliciting labels.

[1] Available at `https://zenodo.org/record/1162952#.W3aQFNgzbRY`

3. The reviewer labels the presented documents and repeats until budget $B = 50$ reviews is exhausted.

Unfortunately, as the dataset does not contain annotations, we just used the keywords present in the title of the corresponding published SLR as corpus-level positive-class-indicating annotations. In other words, we ask the machine to allocate a total share of ρ to the key-words present in documents labeled as relevant; for the irrelevant documents, however, we let the machine decide as we do not have any external knowledge. Although, these key-words significantly overlap (syntactically and semantically) with the search string (see Table 6.2) used to construct the database, the intuition is that the machine will figure out when these terms are relevant and when they aren't. Further, the BM25 bootstrap query is a subset of the key-words, which means that all documents in the bootstrap sample will contain a few of the keywords.

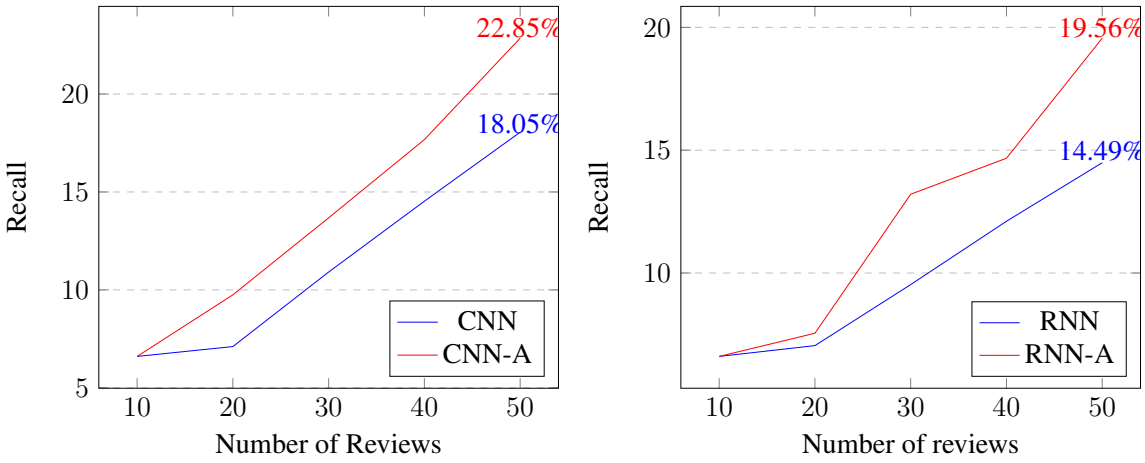

Figure 6.1: Recall Curves for Hall: The graphs show the percentage of relevant documents retrieved over 5 review rounds. The bootstrap sample (first 10 documents) is obtained by using *defect prediction* as the query for BM25 ranking (as suggested by (Yu et al. 2017)). In subsequent rounds, the top 10 most relevant documents in the unlabeled pool, as predicted by the machine, are shown to the user. The curves corresponding to CNN-A and RNN-A are obtained by augmenting the machine with keywords extracted from the title of (Hall et al. 2012): *software,defect,prediction,performance* (used *defect* instead of *fault*, as *fault* isn't present in the bootstrap sample) as relevant-class indicating terms. Interestingly, the machines with annotations retrieve about 5% more relevant documents although there is a large overlap (syntactic/semantic) between bootstrap query, keywords and search string *((fault* OR bug* OR defect* OR errors OR corrections OR corrective OR fix*) in title only AND (Software) anywhere in study)* used for creating the database. The results for other 3 datasets are similar and are presented in Figure 6.2.

As these terms are not completely indicative of the class, we set the share $\rho = 0.5$ to

Paper	#Documents	Relevant Docs
Hall et al. [2012]	8911	102 (1.15%)
Wahono [2015]	7002	62(0.9%)
Radjenović et al. [2013]	6000	48(0.8%)
Kitchenham and Brereton [2013]	1704	45(2.6%)

Table 6.1: Dataset Overview: Approximately only 1-2 % of the documents are relevant.

these terms and set threshold $\tau = 0.4$; although, we did not formally examine the performance for all values of ρ and τ as we got reasonable performance, we note that $\rho = 0.9$ gave poorer results than without annotations. We used the collection of all 4 datasets along with about 20k articles, published in software engineering related conferences, extracted from IEEE Xplore for inducing the domain specific word2vec-based word embeddings. The experiments are repeated 30 times to account for variance and we use the percentage of relevant documents retrieved during the review process as the evaluation metric. Surprisingly, we find that even these annotations are able to help retrieve about 5% more relevant documents (and up to 10% more for Kitchenham) than the machine not augmented with annotations (Figure 6.1). Further, through informal experiments, we find that the irrelevant documents (not included in the SLR) returned by the machine augmented with annotations are qualitatively more relevant than the ones retrieved by the machine not augmented with annotations.

Although we did not perform hyperparameter optimization, we do not expect the Neural Nets augmented with these simple annotations to significantly outperform existing state-of-the-art approaches with SVMs (Yu et al. [2016]). The results obtained in the text classification task and the improvements stemming from the use of simple annotations provide some evidence that there is potential in exploiting unsupervised pre-trained embeddings for the task of primary study selection, which is also a kind of text classification, by the use of more useful corpus-level or document specific annotations. Although most of the keywords exist in all of the bootstrap instances and are not truly indicative of the relevant-class, the neural net can still make use of the annotations by taking into account the context and word order in determining when the keywords are indicative of the class.

Hall	**Query**:	defect prediction
	Bootstrap:	7 Relevant + 3 Irrelevant
	Keywords:	software, defect, prediction, performance
	Search String:	(fault* OR bug* OR defect* OR errors OR corrections OR corrective OR fix*) in title only AND (Software) anywhere in study
Wahono	**Query**:	defect prediction
	Bootstrap:	6 Relevant+4 Irrelevant
	Keywords:	software, defect, prediction, dataset, framework
	Search String:	(software OR applicati* OR systems) AND (fault* OR defect* OR quality OR error-prone) AND (predict* OR prone* OR probability OR assess* OR detect* OR estimat* OR classificat*)
Radjenovic	**Query**:	defect prediction metrics
	Bootstrap:	4 Relevant+6 Irrelevant
	Keywords:	software, defect, prediction, metrics
	Search String:	software AND (metric* OR measurement*) AND (fault* OR defect* OR quality OR error-prone) AND (predict* OR prone* OR probability OR assess* OR detect* OR estimat* OR classificat*)
Kitchenham	**Query**:	systematic literature review
	Bootstrap:	2 Relevant+8 Irrelevant
	Keywords:	systematic, literature, review
	Search String:	any paper doing literature review and published in selected Software Engineering avenues

Table 6.2: Although the query is a subset of the keywords in all of the datasets and the keywords significantly overlap with the search strings used to collect candidate studies, we report improvements in recall up to 5%. The Queries are the query terms (adapted from Yu and Menzies [2017]) used for scoring the documents using BM25. The Bootstrap constitution is as indicated.

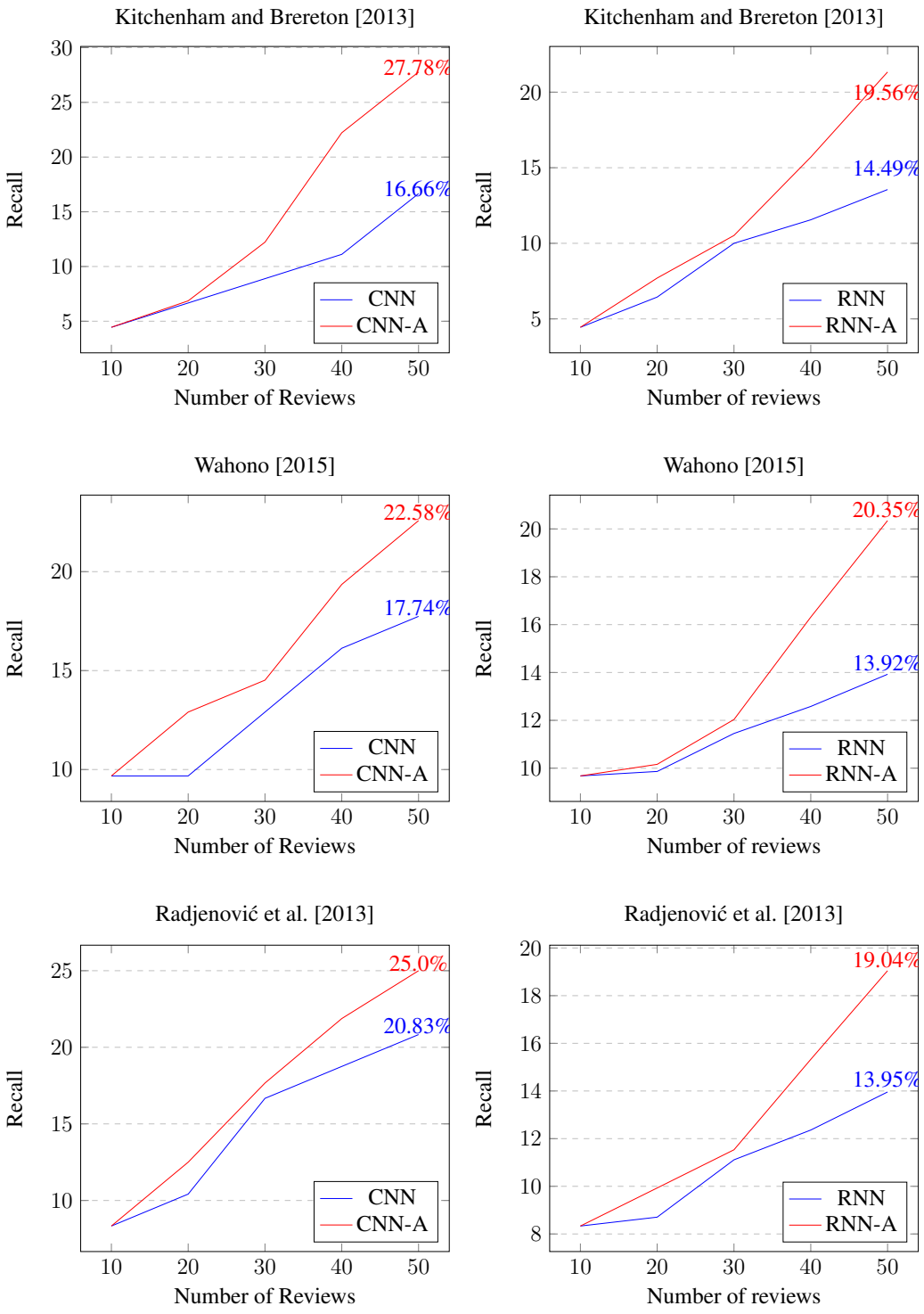

Figure 6.2: Recall Curves for Kitchenham, Wahono and Radjenovic: The graphs show the percentage of relevant documents retrieved over 5 review rounds. The first 10 documents are retrieved using BM25 scoring based on the query terms (Table 3). In subsequent rounds, the top 10 most relevant documents in the unlabeled pool, as predicted by the machine, are shown to the user. The curves corresponding to CNN-A and RNN-A are obtained by training the machine with the corresponding keywords (see Table 3) as corpus-level annotations. Interestingly, the machines with annotations retrieve about 5% more relevant documents although all the documents in the corpus have at least one of these terms.

Chapter 7

Discussion and Future Work

In this work, we demonstrated how annotations and gradient based attributions can be used for better exploiting unsupervisedly learnt word embeddings in text classification and information retrieval. Further, we demonstrated the performance in three different kinds of application scenarios: 1) all documents fully annotated, 2) only some documents fully annotated and 3) some documents partly annotated.

Deep Embeddings While word embeddings support inductive transfer at just the model's first layer, inductive transfer across weights of the model, as demonstrated in ULMFiT(Howard and Ruder [2018]), have not only achieved better state-of-the-art accuracies, but also have demonstrated good performance on low-shot accuracies; specifically, for the Maas et al. [2011] dataset, they state that 10x more documents are needed to match accuracies of 100 labeled documents trained on pre-trained language model. It would be useful to check how annotations work in conjunction with transfer-learning models. The drawback of ULMFiT, however, is that it seems to be designed for RNNs only.

Few-shot Learning Few-shot text classification aims to induce a classifier from a training dataset that consists of fewer examples per class. In the area of few-shot text classification, transfer-learning from a number of related tasks has been shown to be useful (Yu et al. [2018]). For example, they transfer knowledge learnt on sentiment classification of reviews of 57 varieties of products to sentiment classification of book reviews. In our work, we exploit annotations for *transferring* knowledge directly from the user onto the system and demonstrate significant improvements. It would be interesting to examine how annotations can be combined with transfer-learning to improve performance on few-shot text classification. Empirically, we examined how user-annotations could be used for 10-way 1-shot classification on the Yahoo Answers dataset. For this experiment, we randomly chose one document from each class and manually annotated the document. We found that using annotations, even without any transfer learning, can boost the accuracy from around

15% to about 31%!

Other Possible Future Directions For Information Retrieval, we observed that the retrieved documents under dual-supervised settings are qualitatively more relevant than the ones trained without dual-supervision. It would be nice to investigate the qualitative and quantitative benefits when information retrieval is done with dual supervision. While the above directions consider the text classification or information retrieval task, a more interesting research direction is to explore how additional user knowledge can help in more challenging tasks like Question-Answering systems, Text Translation and Image-captioning. Further, the additional knowledge need not always come from the user; it can even be done in an unsupervised manner.

Conclusion In this work, we show how we can factor in user annotations, in the form of highlighted phrases indicative of the class, using gradient-based attributions for training CNNs and RNNs. Our findings indicate that the pre-trained embeddings induced in an unsupervised manner can be best exploited when coupled with annotations. We find that this arrangement can help us achieve accuracies of up to 70% for as few as 10 labeled and annotated instances; something not heard of previously. More importantly, we achieve these results with the commonly used entropy-based query strategy. In information retrieval, we find that using just a few keywords as annotations can help us retrieve more relevant documents not only quantitatively but also qualitatively. As this is a specialized application of text classification, we believe similar improvements can be obtained by exploiting pre-trained embeddings better by the use of more useful annotations. We intend to investigate the quantitative and qualitative benefits in our own future work.

Bibliography

Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations (ICLR 2018)*, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

Gordon V Cormack and Maura R Grossman. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 153–162. ACM, 2014.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

Isabelle Guyon, Gavin Cawley, Gideon Dror, and Vincent Lemaire. Results of the active learning challenge. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, pages 19–45, 2011.

Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 328–339, 2018.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and software technology*, 55(12):2049–2075, 2013.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association

for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Prem Melville and Vikas Sindhwani. Active dual supervision: Reducing the cost of annotating examples and features. In *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*, pages 49–57. Association for Computational Linguistics, 2009.

Prem Melville, Wojciech Gryc, and Richard D Lawrence. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1275–1284. ACM, 2009.

Stephen Merity, Bryan McCann, and Richard Socher. Revisiting activation regularization for language rnns. *arXiv preprint arXiv:1708.01009*, 2017.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

Andrew Slavin Ross, Michael C Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. *arXiv preprint arXiv:1703.03717*, 2017.

Manali Sharma, Di Zhuang, and Mustafa Bilgic. Active learning with rationales for text classification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 441–451, 2015.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *CoRR*, abs/1605.01713, 2016. URL http://arxiv.org/abs/1605.01713.

Kevin Small, Byron C Wallace, Carla E Brodley, and Thomas A Trikalinos. The constrained weight space svm: learning with ranked features. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 865–872. Omnipress, 2011.

Romi Satria Wahono. A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks. *Journal of Software Engineering*, 1(1):1–16, 2015.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. Diverse few-shot text classification with multiple metrics. *arXiv preprint arXiv:1805.07513*, 2018.

Zhe Yu and Tim Menzies. Fast^2: an intelligent assistant for finding relevant papers. *CoRR*, abs/1705.05420, 2017. URL `http://arxiv.org/abs/1705.05420`.

Zhe Yu, Nicholas A Kraft, and Tim Menzies. Finding better active learners for faster literature reviews. *arXiv preprint arXiv:1612.03224*, 2016.

Omar Zaidan, Jason Eisner, and Christine Piatko. Using annotator rationales to improve machine learning for text categorization. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 260–267, 2007.

Omar F Zaidan and Jason Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 31–40. Association for Computational Linguistics, 2008.

Jiacheng Zhang, Yang Liu, Huanbo Luan, Jingfang Xu, and Maosong Sun. Prior knowledge integration for neural machine translation using posterior regularization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1514–1523, 2017a.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

Ye Zhang, Iain Marshall, and Byron C Wallace. Rationale-augmented convolutional neural networks for text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2016, page 795. NIH Public Access, 2016.

Ye Zhang, Matthew Lease, and Byron C Wallace. Active discriminative text representation learning. In *AAAI*, pages 3386–3392, 2017b.

Appendix A

Architecture-specific adaptations of Misattribution Error

A.1 CNNs

In the CNN architecture considered, the misattribution error is hard to optimize because the gradients through the 1-max-pooling will contain zeros and will not be informative enough. Essentially, the machine should first suppress values from non-useful features and then wait around till the right values are chosen by the machine. This increases train time. Further, another challenge is that if the machine does end up selecting *most* of the user-suggested features, the zero gradients to the non-user-suggested features does not really guarantee that the activation values of the non-user-suggested features are sufficiently suppressed. This makes the optimization hard.

Let's call the given CNN network defined in the main paper as N. We propose computing the attributions for the misattribution error by considering a proxy network P instantiated with the exact same weights as N but with 1-max pooling replaced with sum; i.e, the sentence features are computed by summing across all of the feature maps instead of computing max. This enables the machine to adjust attributions of all words, in parallel. Let $\tilde{A}^P$ be the attributions computed using the network P.

$$Misattribution_{Error} = \mu_c(A^U, \tilde{A}^P) \tag{A.1}$$

However, there is one problem with this approach: the machine gets the same reward irrespective of whether it chooses to minimize the objective function by using several filters or just a few filters. In order to better demonstrate, consider the hypothetical feature maps shown in Table A.1 and A.2; the shaded columns correspond to the activation values from the user-suggested features. Observe that both of them satisfy the objective function in Eq A.1, although, the misattribution error for the true network N would be *truly* minimized if the feature maps looked like Table 2. Another side-effect if the machine chose to behave like the one showed in Table 1 is that the machine ends up under-utilizing the number of

filters and reduces the capacity. Note that this is just one of the many possible ways to not result in desired attribution distribution.

	1	2	3	4	5
A	0.5	2.5	2.75	0.01	0.2
B	0.5	0.01	-1.00	0.05	0.8
C	0.01	0.03	-0.2	0.05	0.01

Table A.1: Hypothetical feature maps after training with Eq. 1 when 3 filters A,B and C of same region size are used. Note that each activation corresponds to a set of words. Greater the activation value, greater the attribution value.

	1	2	3	4	5
A	0.5	2.5	1.2	0.01	0.2
B	0.5	0.01	2.75	0.05	0.8
C	0.01	0.03	-0.2	0.05	0.01

Table A.2: Hypothetical feature maps after training with Eq. 1 when 3 filters A,B and C are used. Note that each activation corresponds to a set of words.

The fix to this problem would be to introduce an additional term in the objective function encouraging the machine to optimize Eq A.1 by using features across different filters instead of features across the same filter. Note that we are primarily concerned in maximizing the number of filters which select user-suggested features when 1-max pooling is applied. Let us first consider the non-zero attributions to the user-suggested features from each of the filters such that each filter is allowed to transfer information from only one set of input words; these can be extracted greedily by applying 1-max-pooling only to activations corresponding to user-suggested features (Ex: along A3, B2 and C2 for feature maps in Table 1). It is easy to see that pushing these attribution scores higher would encourage the proxy network P to use features across many filters. Let $A_i^{B,c}$ be the attribution assigned to i^{th} user-suggested feature along the best-possible path leading to logit of class c. We choose to maximize the sum of these attributions $\sigma_c = \sum_i A_i^{B,c}$ as we do not know the exact distribution of attribution among user-defined features; however, as we do not know the required magnitude of these attributions, we need to maximize it in comparison to another number. For simplicity, we choose to maximize the attributions with respect to the expected class E in comparison to the other classes. Let's quantify the total attribution from user-specified features leading up to expected class, E, with respect to other classes using softmax as γ_E:

$$\gamma_E = \frac{\exp \sigma_E}{\sum_c \exp \sigma_c} \tag{A.2}$$

Finally, we can define the misattribution error using cross-entropy loss as:

$$Misattribution_{Error} = \mu_c(A_i^U, \tilde{A}_i^P) + log(\gamma_E) \tag{A.3}$$

Interestingly, if the activation used is ReLU or Leaky-ReLU, the second term is equivalent to adding a new example having only user-labeled features. Thus, masking features for factoring in user-suggestions, like in traditional methods, is applicable in certain cases such as this and is a special case of attribution; however, unlike in traditional models, this is similar to *adding* a new example rather than modifying an *existing* data point.

In the text classification task: If $\mu_c(A_i^U, \tilde{A}_i^P)$ is used alone, we observe that the accuracies with training pool smaller than 50 examples remain unchanged, but saturates way earlier as the machine starts using fewer filters and has a reduced capacity as illustrated. Further, if the machine is trained with just $log(\gamma_E)$, the accuracies with larger number (≥ 100) of training examples remain relatively unchanged but those with fewer examples have a lesser accuracy; this is because the machine does not get automatically tuned to ignore non-annotated words when there are fewer examples. In the information retrieval task, however, we did not notice any advantage (even for higher number of training examples) of using the second term as the number of user-suggested features were very few.

A.2 RNNs

For the same reasons as in max-pooling, optimizing through the attention is hard because the gradients are too small for the non-attended features. As we chose to compute attributions by setting F to class logits in CNNs, we choose to optimize using feature vector v_s (Eq 2 in main paper) in RNNs. Following the same idea as in CNNs, we propose to compute attributions using hidden states without attention (i.e. h_i instead of $\alpha_i h_i$) and independently optimize the α_is to select user-suggested features. In order to optimize attentions, we can easily write the misattribution error by equating the normalized attributions $\tilde{A}_i$ with the attentions α_i:

$$Misattribution_{Error} = \mu_f \left(A_i^U, \alpha_i\right) \tag{A.4}$$

We need to now factor user suggestions into the computation of hidden features: in order to do that, we can very easily compute the attributions by setting F to be the sequence

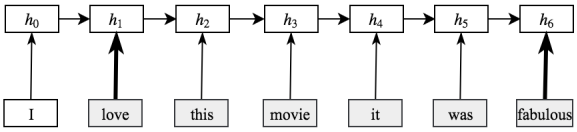

Figure A.1: RNN (without attentions and softmax) with an example sentence. The annotated words (shaded) can be interpreted as those words worth accumulating in the hidden state. The thickened arrows indicate those terms which the machine might remember.

of hidden states; in other words, the gradients from all hidden states at time steps $\geq t$ would be considered to compute attribution of word w_t as shown in Eq A.5 [the derivatives are computed at $\mathbf{V_0} = \mathbf{w_0}, \mathbf{V_1} = \mathbf{w_1} ... \mathbf{V_n} = \mathbf{w_n}$]

$$A_i = \mathbf{w_i} \cdot \left(\frac{\partial \mathbf{h_i}}{\partial \mathbf{V_i}} + \frac{\partial \mathbf{h_{i+1}}}{\partial \mathbf{V_i}} + \cdots + \frac{\partial \mathbf{h_n}}{\partial \mathbf{V_i}} \right) \tag{A.5}$$

Using this, we can write out the complete misattribution error as:

$$Misattribution_{Error} = \mu_f \left(A_i^U, \alpha_i \right) + \mu_f \left(A_i^U, \tilde{A}_i \right) \tag{A.6}$$

However, this is computationally expensive and we propose approximating it by just considering the gradients from hidden state at time $t = i$. Turns out that this approximation works quite well not only for those cases where all the hidden states are used but also where the last hidden state is used. Mathematically, if A_i is large, $\mathbf{w_i} \cdot \frac{\partial \mathbf{h_i}}{\partial \mathbf{V_i}}$ should be large as well; this is because, if the gradient of hidden state at i^{th} time step with respect to w_i is small, the gradients of hidden states at later time steps would be smaller as they are a function of h_i.

Gated Recurrent Neural Networks can be interpreted as machines which read through a sequence of inputs and accumulate useful information in the hidden states from the inputs that it finds important (Fig. A.1). From the figure, it is easy to see how the approximation satisfies attribution with respect to both feature vectors: mean or weighted mean of all hidden states and last hidden state. Note that this is just an approximation to speed up the training for single recurrent layer.

A.3 Hyperparameters: ρ and τ

For the text classification task, we find that the accuracies are similar for $0.8 \leq \rho \leq 0.9$ and $0.7 \leq \tau \leq 0.8$. If τ is set to reasonable values ($\tau >= 0.5$) beyond these ranges, the accuracies for instances trained from fewer than about 50 examples are negatively affected

as the machine becomes less robust to long annotations, while the others are relatively stable. It is useful to keep the ρ value < 1 and allow the machine to consider the unannotated words in at least 2 cases: when not all of the class-indicating-features are actually annotated and/or for allowing the machine to learn the context for selecting the annotated features. For the information retrieval task, unsurprisingly, we find that setting higher ρ values (> 0.5) reduces the recall.

Appendix B

Additional Text Classification Results

In this section, we illustrate the advantages of combining annotations with unsupervised word embeddings by benchmarking with Sharma et al. [2015]. We simulated the human labeler by extracting features as described in the paper: we choose only the positive (negative) features that had the highest χ^2 (chi-squared) statistic in at least 5% of the positive (negative) documents. Further, the artificial labeler returns any one word, rather than the top word or all the words, as rationale. Here is the description of the datasets:

- **WvsH**: This classification task consists of classifying between comp.os.ms-windows.misc and comp.sys.ibm.pc.hardware in the 20-Newsgroups classification task.
- **Nova**: This is a binary classification task derived from the 20-newsgroups classification task. More details can be found at Guyon et al. [2011].
- **SRAA**: This is a binary classification task consisting of 48K documents that discuss either auto or aviation.

The improvement in the AUCs are illustrated in Table B.1. We derived the AUCs by taking a mean of the AUCs obtained by RNNs and CNNs when trained on 10 randomly chosen documents - 5 from each class. The AUCs for Sharma et al. [2015] are extracted from the paper.

Dataset	Sharma et al. [2015]	Ours
WvsH	80%	87.25%
Nova	83%	90.53%
SRAA	86%	91.37%

Table B.1: Text classification.

Additionally, we also empirically evaluated the gains we would get when we perform one-shot text classification with annotations using Yahoo! Answer Classification dataset.

In this 10-way classification task, we randomly chose 1 document from each class and manually annotated for the purposes of the experiments. In the traditional Machine Learning method, we obtained an accuracy of about 14% with annotations while we obtained 28% with RNNs and 31% with CNNs.

Appendix C

Useful Insights for Real-world Applications

In this chapter, we will present some insights from the various experiments we conducted which could be useful in training setup for solving real-world problems like Total-recall Information Retrieval. Some of the things include:

1. **Tuning the Architecture**: Typically, a machine tuned to run well on X training examples does not necessarily run well when trained on Y examples when $X << Y$. However, in our experiments, we found that when the machine is tuned to work on X training examples, it will typically work well on $X \pm 100$. A possible research direction could be to dynamically grow the network such that previously trained weights are still retained.

2. **Empirically testing the gain on using annotations**: It is useful to first investigate what kind of annotations are useful by annotating few examples (say, between 10 and 50). In order to better estimate the performance, it is useful to strategically sample a variety of documents instead of just randomly sampling. In our experiments, we found that the length of the text is one of the most important features in distinguishing texts. Other syntactic or semantic features could be used to get better representative samples; these could either be hand-picked or selected by clustering.

3. **Monitoring learning**: As annotations can sometimes have unintended side-effects, it is useful to track progress by investigating the key features the machine is considering in assigning the class. This can serve as a feedback towards what kinds of annotations are most useful. This is a crucial step in cases where annotations aren't as direct as sentiment-bearing words.

4. **Crowd-sourcing annotations**: Some times, it might be useful to crowd-source the annotations for training so as to eliminate bias. Further, the strategy for combining the crowd-sourced annotations is important in setting the share. For example, depending on the portions of annotations ignored, it might make sense to allocate a

lesser share to the annotations. Furthermore, in case all annotations are retained, the parameter ρ can be suitably adjusted to allocate higher shares depending on the votes.

5. Here are examples of annotated positive and negative examples from the sentiment-classification dataset (Pang and Lee [2004]):

   (a) one of my colleagues was surprised when i told her i was willing to see betsy's wedding . and she was shocked to hear that **i actually liked it** . her reaction was understandable when you consider that the film revolves around molly ringwald , who hasn't made a worthwhile film since 1986 . but the fact is , betsy's wedding is also an alan alda film . and while ringwald has been making duds for the last four years , alda has been involved with several noteworthy projects , including crimes and misdemeanors and a new life . written and directed by alda , betsy's wedding is a vibrant slice-of-life , mixing a few dramatic moments into a big bowl of whimsical humor . **alda's comic elixir is smooth and refreshing** –and **a welcome change** of pace from the usual summer fare . as bride and groom , molly ringwald and dylan walsh are the pivotal characters in the film , but they are by far the least interesting . walsh is a nonentity , with all the screen presence of a door knob . ringwald is simply unbearable and is easily the weakest link in the chain . she looks hideous with her short-cropped orange hair , red lip-stick and grotesque outfits . she's supposed to be a dress designer , but she looks more like a clown . and to make matters worse , ringwald's performance matches her appearance . thankfully , alda keeps ringwald's screen time to a minimum ; he is far more interested in the colorful periphery characters . the wedding is just a device to bring together the bride's working-class , italian family and the groom's rich , gentile family . ringwald's folks are homey and down-to-earth , with alda as her free-spirited father , madeline kahn as her practical mother , and ally sheedy as her lonely sister . walsh's clan , on the other hand , is prim , proper and ostentatious . when the two families meet and mingle , the movie becomes a story of culture clash , or as one character puts it , " money versus values . " ally sheedy , in a wonderfully understated performance , is **one of the film's most pleasant surprises** . sheedy expresses more with just her eyes than ringwald does with her entire

body . it's anthony lapaglia , however , who seizes the spotlight . lapaglia plays stevie dee , a suave , overly polite mafioso who is formally courting sheedy with old-fashioned chivalry . lapaglia's sincere but dim-witted character **is a riot** . and what's uncanny is that lapaglia is a dead ringer for robert de niro , with a little bit of alec baldwin thrown in for good measure . lapaglia seems to have attended the de niro school of gangster acting , and his **inspired performance** is partly a tribute to his role-model and partly a rip-off . i don't know whether to say a star is born or a star is re-born , but i do know that lapaglia's **over-the-top performance** should not be missed . the scrumptious comic acting , however , **extends well** beyond sheedy and lapaglia . joe pesci , in particular , sinks his teeth into his role as alda's unscrupulous brother-in-law , a slum lord with mob ties , who is cheating on his wife (catherine o'hara) . alda , faced with challenge of both directing and acting , somehow **finds just the right comic touch** as the bride's financially-strapped father , a carpenter whose dreams are bigger than his wallet . the film adopts alda's psychological point of view as he tries to one : plan the wedding , and two : pay for it . as a filmmaker , alda's style of humor is remarkably restrained and tasteful . and while he doesn't have the comic genius of a woody allen , **alda does possess the inspiration** to make movies which are **ten times more entertaining** than the slop which usually passes for comedy .

(b) best remembered for his understated performance as dr . hannibal lecter in michael mann's forensics thriller , manhunter , scottish character actor brian cox brings something special to every movie he works on . usually playing a bit role in some studio schlock (he dies halfway through the long kiss goodnight) , he's only occasionally given something meaty and substantial to do . if you want to see some brilliant acting , check out his work as a dogged police inspector opposite frances mcdormand in ken loach's hidden agenda . cox plays the role of big john harrigan in the disturbing new indie flick l . i . e . , which lot 47 picked up at sundance when other distributors were scared to budge . big john feels the love that dares not speak its name , but he expresses it through seeking out adolescents and bringing them back to his pad . what bothered some audience members was the presentation of big john in an oddly empathetic light

. he's an even-tempered , funny , robust old man who actually listens to the kids' problems (as opposed to their parents and friends , both caught up in the high-wire act of their own confused lives .) he'll have sex-for-pay with them only after an elaborate courtship , charming them with temptations from the grown-up world . l . i . e . stands for long island expressway , which slices through the strip malls and middle-class homes of suburbia . filmmaker michael cuesta uses it as a (pretty transparent) metaphor of dangerous escape for his 15-year old protagonist , howie (paul franklin dano) . in his opening voice-over , howie reveals a morbid preoccupation with death on the road , citing the l . i . e . highway deaths of filmmaker alan j . pakula , songwriter harry chapin , and his own mother on exit 52 . he's both fascinated and disturbed by the l . i . e . , and those feelings are projected onto big john (who follows howie around in his bright red car , but never makes a move to force the boy to do something he doesn't want to do . this makes him much more complex than the usual child molesters seen in movies – he's a beast , but ashamed of it .) l . i . e . would have worked best as a half-hour short film about howie's ill-advised foray into big john's haven . **there is unnecessary padding** with howie's miserable dad (bruce altman) in the hot seat for a white-collar crime , degenerate youngsters who get their kicks from robbing middle-class houses , and some homoerotic shenanigans with wise-ass gary terrio (billy kay) , a handsome artful dodger . rather than add to the themes of suburban ennui (not that we needed another movie on that subject) , **these awkward subplots pad out the running time** to adequate feature length . concurrently , the relationship between howie and big john is evenly paced and exceptionally well acted . cox , sporting a baseball cap and a faded marine tattoo , is all bluff and bluster . dano is quiet and at first glance seems so withdrawn as to be transparent . we're so used to child actors whose dramatic choices are broad and obvious (calling haley joel !) , it's surprising to see one who actually listens throughout any given scene . the restraint is admirable . but l . i . e . 's screenplay doesn't always give them the best material . when howie reads big john a walt whitman poem , the moment feels a bit too precious . director michael cuesta lingers on an ecstatic reaction shot of big john , who may as well be hearing glenn gould performing bach's

goldberg variations . **it's too much** . there are also some obvious dramatic contrivances involving big john's other boy toy (walter masterson) , jealous over the newbie . this plot thread predictably leads to violence . not content to be a haunting , observational portrait of teen alienation in a royally screwed up world (like terry zwigoff's superb ghost world) , cuesta lacks the confidence in his own work to end on an ambivalent note . it's typical of **unimaginative cinema to wrap things up with a bullet** , sparing the writers from actually having to come up with a complex , philosophical note . in this regard , l . i . e . (and countless other indie films) share something in common with blockbuster action films : problems are solved when the obstacle is removed . how often does real life work this way ? to extend the question : if a movie is striving for realism , do dramatic contrivances destroy the illusion ?

Appendix D

Programming Details

D.1 Libraries used

All the code is written in python 2.7 (with __future__ import to allow for compatibility with Python3). The following libraries are used in the project:

1. Tensorflow - We used version 1.7 in this project for the experiments. Although, a newer version is released almost every month, we expect the code to be compatible with future releases. We used the Tensorflow for building the CNN.

2. PyTorch - We used version 0.4. We used PyTorch for building the RNN as Tensorflow cannot compute second derivatives.

3. Numpy - We used numpy for common matrix operations.

4. NLTK - We used NLTK for stopwords and for tokenizing documents.

5. Gensim - We used version 3.3.0. We used this for training word embeddings and loading pre-trained embeddings.

D.2 Readme

The exact details of how the program needs to be run is mentioned in the GitHub repo: `https://github.com/chandramouli-sastry/dual-AL`. There are essentially 5 kinds of classes as described below. The GitHub Repository has 4 Python Packages – 1shot, IR, classification and featExpert_AL; each of these packages have their own specialized versions of these 5 classes.

1. driver - This is the main program that reads the data in and creates the list of Document objects, which is used by the ActiveLearning class. For the text classification task, each of the documents are in separate files and grouped into folders according

to their class. For the information retrieval task, all the documents and their labels are part of one csv file. The driver takes care of abstracting out the storage details and creates a list of Document objects.

2. Document - This file contains the definition of Document class. Each document has an id, title (if present), text and class label as features. Further, the annotated features (if present) are also included. It is responsible for any pre-processing - which as of now is just word tokenization.

3. RNN-Factory - This file contains the definition of RNN-Factory class which can return two kinds of classes based on the parameters : one with dual supervision and one without dual supervision. These classes are used to instantiate the classifiers used by the ActiveLearning class. This is implemented in PyTorch.

4. CNN-Factory - This is highly similar to the RNN-Factory class excepting that it internally constructs a CNN architecture and is written in Tensorflow.

5. ActiveLearning - This class is instantiated with the training pool [a list of Document objects], a held-out test data [a list of Document objects], the classifier object [CNN/RNN] and the budget [in terms of the maximum number of documents to select from the traning pool].